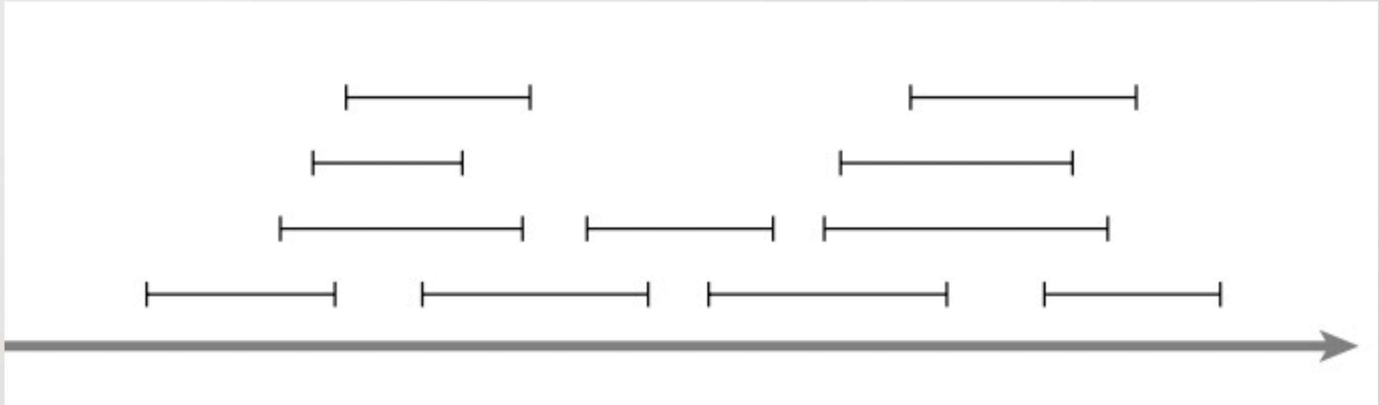
Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

- Temos um recurso, por exemplo, uma sala de aula, laboratório, vaga no estacionamento coberto... e muitas pessoas solicitam usar o recurso por um período de tempo.
- Uma requisição possuí a seguinte forma: Posso utilizar o recurso começando no tempo s até o tempo f?
- Assuma que cada recurso pode ser utilizado por apenas uma pessoa de cada vez.
- Um escalonador deseja aceitar um subconjunto dessas solicitações, rejeitando todas as outras, de forma que as solicitações aceitas não se sobreponham.
- **O objetivo é maximizar o número de requisições aceitas!**

Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

O ALGORITMO:

- Formalmente, teremos n requisições rotuladas de $1, \ldots, n$ com cada requisição i especificando o tempo de início s_i e término f_i do uso.
- O conjunto de todas as solicitações i é chamado de R.
- Onde sempre teremos $s_i < f_i$ para as requisições i.
- Duas requisições são *compatíveis* se os intervalos solicitados não se sobreporem: ou seja, se a solicitação i iniciou antes da solicitação j, teremos $f_i \leq s_j$, caso tenha ocorrido após j teremos $f_j \leq s_i$.
- Dizemos que o subconjunto de solicitações A é compatível ser para todos os pares $i, j \in A, i \neq j$ são compatíveis.
- O objetivo do algoritmos greedy é construir o subconjunto A com o maior número de requisições, ou seja, $\max(|A|)$.

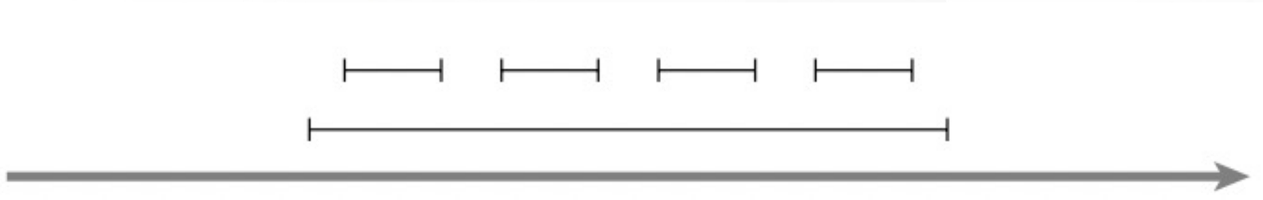


Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

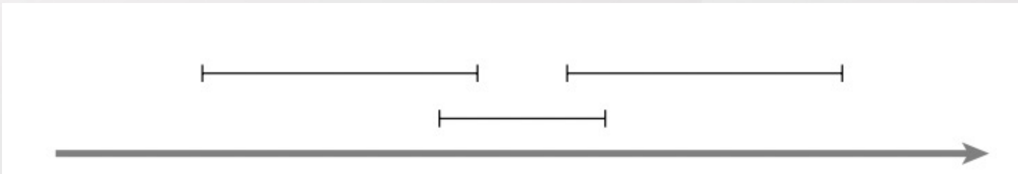
- A ideia básica do algoritmo greedy para o problema de escalonamento é utilizar uma regra simples para selecionar a primeira solicitação i_1.
- Uma vez que a solicitação i_1 é aceita, rejeitamos todas as solicitações que não são compatíveis com i_1.
- Procedemos então para a próxima solicitação i_2, e novamente rejeitamos todas a solicitações que não compatíveis com i_2.
- Procedemos desta forma até não existirem mais solicitações em R.
- O desafio é qual regra simples de decisão selecionar:
  - Selecionar a requisição que começa mais cedo, ou seja, o menor $s(i)$?
  - Selecionar a requisição com o menor tempo de duração $\big(f(i) - s(i)\big)$?
  - Selecionar a requisição com o menor número de conflitos/sobreposições?
  - Selecionar a requisição que termina mais cedo, o menor $f(i)$?

Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

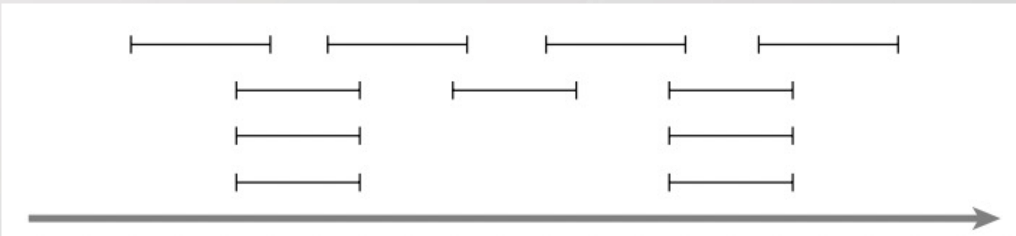
- Selecionar a requisição que começa mais cedo, ou seja, o menor $s(i)$?



- Selecionar a requisição com o menor tempo de duração $(f(i) - s(i))$?



- Selecionar a requisição com o menor número de conflitos/sobreposições?



Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

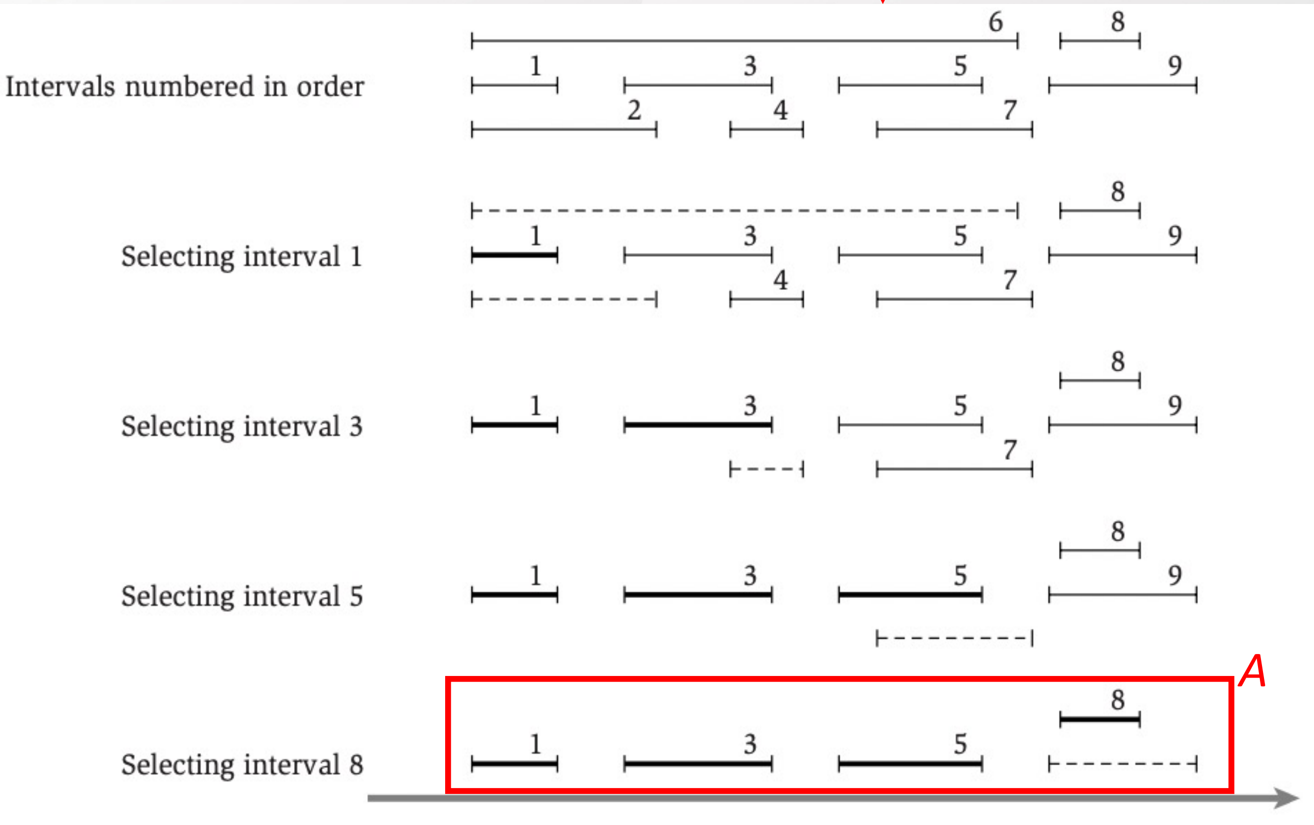
- Selecionar a requisição que termina mais cedo, o menor $f(i)$?
- Essa é a regra greedy que leva a solução ótima (maior número de requisições aceitas) para este problema.
- Neste caso aceitamos as requisições que possuem o menor $f(i)$.
- Esse ideia é bem natural, pois garantimos que nossos recursos estão disponíveis o mais rápido possível, ao mesmo tempo que satisfazemos as requisições.
- Desta forma maximizamos o tempo restante para satisfazer outras solicitações.

```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i∈R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i ∈ R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

O conjunto ótimo ($\mathcal{O}$) para este problema possui 4 intervalos, ou seja, $|\mathcal{O}| = 4$.



O conjunto *A* proposto pelo algoritmo greedy é o único subconjunto ótimo?

ANALISANDO O ALGORITMO:

- O método greedy é muito intuito e natural, porém não é obvio que ele retorna o conjunto ótimo de requisições de intervalos $\mathcal{O}$.
- $\mathcal{O}$ é definido com o conjunto com o maior número possível de requisições compatíveis!
- As ideias que sugerimos anteriormente para a regra de decisão do algoritmo greedy também pareciam promissoras. Portanto, precisamos provar matematicamente que o algoritmo greedy selecionado é de fato ótimo.
- Ou seja, queremos provar que $|A| = |\mathcal{O}|$, e não que $A = \mathcal{O}$, o que seria um problema mais complexo dado que podem existir múltiplos $\mathcal{O}$. Para nosso critério é suficiente que $A\ e\ \mathcal{O}$ possuam a mesma quantidade de requisições.
- Começamos declarando que os intervalos da requisições do subconjunto A retornado pelo algoritmo são *compatíveis.*

Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

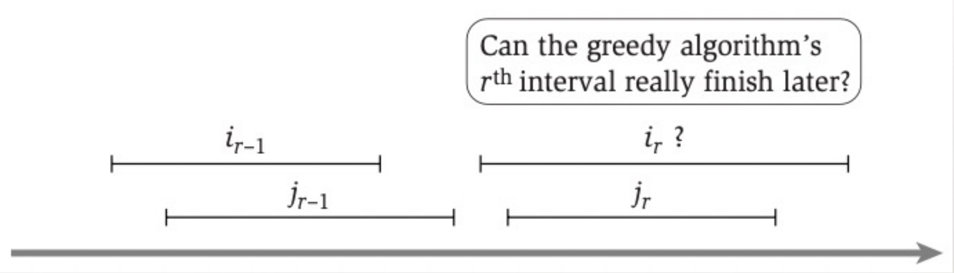
A1: Começamos afirmado que A é um subconjunto de requisições compatíveis, pela forma como o algoritmo opera.

- Suponha que $i_1, \ldots, i_k$ é o conjunto de solicitações em A na ordem em que elas foram adicionadas em A. Note que $|A| = k$.
- Similarmente $j_1, \ldots, j_m$ é o conjunto ótimo de solicitações em $\mathcal{O}$ na ordem em que elas foram adicionadas em $\mathcal{O}$. Novamente $|\mathcal{O}| = m$.
- Nosso objetivo é provar que $k = m$.
- Por definição as requisições em $\mathcal{O}$ são compatíveis, o que significa que os pontos de partida possuem a mesma ordem dos pontos de finalização.
- Nossa intuição é que o algoritmo greedy deseja liberar os recursos o mais rápido o possível assim que a primeira requisição é satisfeita.
- Nossa regra greedy sempre vai garantir que $f(i_1) \leq f(j_1)$, ou seja, ela sempre estará "a frente" do conjunto $\mathcal{O}$. Com requisições terminando antes ou concomitantemente.
- Conseguimos provar isso para a requisição 1, $r = 1$, precisamos provar para $r > 1$.

Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

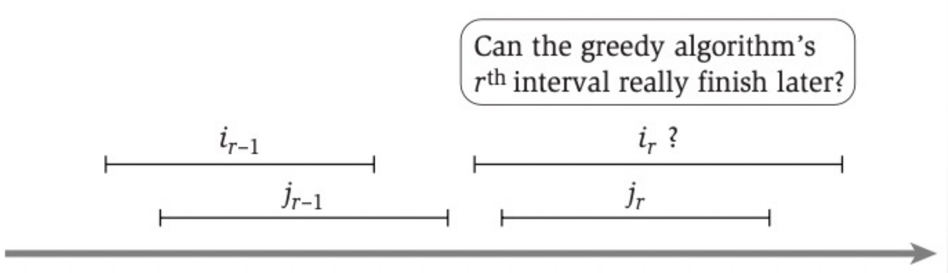
A2: Para todos os índices $r \leq k$ temos $f(i_r) \leq f(j_r)$

- Vamos provar essa parte por indução.
- Já provamos que $r = 1$ a afirmação é verdadeira.
- Agora vamos provar para $r > 1$.
- Vamos assumir como hipótese introdutória que a afirmação é verdadeira para $r - 1$ e vamos provar para r.



- Como demonstrado na figura acima, a **hipótese introdutória** assume que $f(i_{r-1}) \leq f(j_{r-1})$.
- Para que o r^{th} intervalo do algoritmo não finalizar antecipadamente da mesma forma que $r - 1$ ele precisaria ficar atrasado como demonstrado na figura, i_r termina mais tarde.
- Mas existe uma razão para isto não poder acontecer: ao invés de escolher o intervalo que acaba mais tarde, o algoritmo greedy sempre escolhe a opção que acaba mais cedo e acabaria por escolher j_r e não i_r e isso satisfaz o passo de indução.

A2: Para todos os índices $r \leq k$ temos $f(i_r) \leq f(j_r)$



- Mais precisamente, sabemos que $f(j_{r-1}) \leq s(j_r)$, combinando a hipótese de indução $f(i_{r-1}) \leq f(j_{r-1})$, temos que $f(i_{r-1}) \leq s(j_r)$.
- Assim, o intervalo j_r está no conjunto de requisições R ao mesmo temo que o algoritmo greedy seleciona i_r.
- O algoritmo greedy sempre seleciona a requisição com o menor tempo de finalização, como j_r é um desses intervalos, nós temos que $f(i_r) \leq f(j_r)$.
- Isto completa a etapa de indução.
- E portanto, concluímos que o algoritmo greedy sempre antecipa o intervalo $\mathcal{O}$: para cada r, o r^{th} intervalo selecionado finaliza pelo menos tão cedo quanto o r^{th} em $\mathcal{O}$.
- Esta conclusão é utilizada para implicar que o subconjunto A é ótimo.

A3: O algoritmo greedy retorna um subconjunto ótimo A.

- Vamos provar isso por contradição.
- Se A não é ótimo, então o conjunto ótimo $\mathcal{O}$ deve possuir mais requisições, ou seja, $\mathrm{m} > k$.
- Aplicando A.2 com $r = k$, temos que $f(i_k) \leq f(j_k)$.
- Como $\mathrm{m} > k$, existe uma requisição j_{k+1} em $\mathcal{O}$.
- Essa requisição começa após j_k terminar, então i_k termina.
- Portanto, após deletar todas as requisições possíveis em R ainda existe a requisição j_{k+1}.
- Mas o algoritmo greedy termina com a requisição i_k , porém ele só pode finalizar quando R está vazio, o que é uma contradição!!!

PUCRS ESCOLA POLITÉCNICA

Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

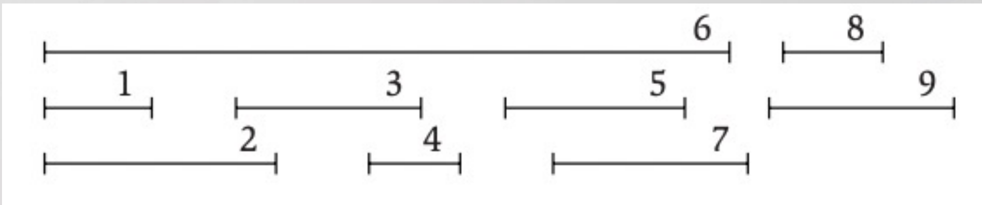
EXERCÍCIO:

Implemente o algoritmo de escalonamento de tarefas!!!!

```
Initially let R be the set of all requests, and let A be empty
While R is not yet empty
  Choose a request i ∈ R that has the smallest finishing time
  Add request i to A
  Delete all requests from R that are not compatible with request i
EndWhile
Return the set A as the set of accepted requests
```

E ordene as seguintes requisições:

```
Request[] requests = {
    new Request(start: 0, finish: 2, name: "R1"),
    new Request(start: 0, finish: 3, name: "R2"),
    new Request(start: 2, finish: 5, name: "R3"),
    new Request(start: 4, finish: 6, name: "R4"),
    new Request(start: 7, finish: 9, name: "R5"),
    new Request(start: 0, finish: 10, name: "R6"),
    new Request(start: 8, finish: 11, name: "R7"),
    new Request(start: 13, finish: 14, name: "R8"),
    new Request(start: 12, finish: 15, name: "R9")
};
```



Algoritmos Gulosos (Greedy): Escalonamento de Tarefas.

<u>IMPLEMENTAÇÃO E TEMPO DE EXECUÇÃO:</u>

- Podemos fazer o algoritmo rodar em $O(n \log n)$.
- Começamos ordenando as n requisições pelo tempo de finalização e marcando cada uma nesta ordem, ou seja, assumimos que $f(i) \leq f(k)$, quando $i \leq k$. Essa ordenação leva o tempo $O(n \log n)$.
- Com um tempo adicional de $O(n)$, construímos um arranjo $S[1 \dots n]$ com a propriedade que $S[i]$ contém o valor de $\mathrm{s}(i)$.
- Depois selecionamos as requisições processando os intervalos na ordem crescente de $f(i)$.
- Sempre selecionamos o primeiro intervalo, e iteramos pelos intervalos até obter o primeiro intervalo compatível, onde $\mathrm{s}(j) \geq f(1)$. E também selecionamos este intervalo.
- De forma geral, se o intervalo mais recente selecionado termina no tempo f, continuamos a iterar pelo intervalo até obter o primeiro j para o qual $\mathrm{s}(j) \geq f$.
- Desta forma implementamos o algoritmo greedy com apenas uma passagem por todos os intervalos em tempo $O(n)$.
- Ou seja, a complexidade do algoritmo é $O(n \log n + n) \Rightarrow O(n \log n)$.

EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish times and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

$S \leftarrow \varnothing$. ← set of jobs selected

FOR $j = 1$ TO n

  IF (job j is compatible with S)

    $S \leftarrow S \cup \{ j \}$.

RETURN S.